Learning Significant Fourier Coefficients over Finite Abelian Groups (2003; Akavia, Goldwasser, Safra)

Adi Akavia, Massachusetts Institute of Technology, theory.csail.mit.edu/~akavia

INDEX TERMS: Fourier transform, Fourier analysis, signal processing, approximation theory, cryptography, coding theory, error correcting codes, computational learning theory, algorithms, compression.

1 PROBLEM DEFINITION

Fourier transform is among the most widely used tools in computer science. Computing the Fourier transform of a signal of length N may be done in time $\Theta(N \log N)$ using the Fast Fourier Transform (FFT) algorithm. This time bound clearly cannot be improved below $\Theta(N)$, because the output itself is of length N. Nonetheless, it turns out that in many applications it suffices to find only the *significant Fourier coefficients*, *i.e.*, Fourier coefficients occupying, say, at least 1% of the energy of the signal. This motivates the problem discussed in this entry: the problem of efficiently finding and approximating the significant Fourier coefficients of a given signal (SFT, in short). A naive solution for SFT is to first compute the entire Fourier transform of the given signal and then to output only the significant Fourier coefficients; thus yielding no complexity improvement over algorithms computing the entire Fourier transform. In contrast, SFT can be solved far more efficiently in running time $\widetilde{\Theta}(\log N)$ and while reading at most $\widetilde{\Theta}(\log N)$ out of the N signal's entries [2]. This fast algorithm for SFT opens the way to applications taken from diverse areas including computational learning, error correcting codes, cryptography and algorithms.

We now formally define the SFT problem, restricting our attention to discrete signals. We use functional notation where a signal is a function $f\colon G \to \mathbb{C}$ over a finite abelian group G, its energy is $\|f\|_2^2 \stackrel{def}{=} \frac{1}{|G|}\sum_{x\in G} f(x)^2$, and its maximal amplitude is $\|f\|_\infty \stackrel{def}{=} \max\{\,|f(x)| \mid x \in G\}$.[1] For ease of presentation we assume without loss of generality that $G = \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \times \ldots \times \mathbb{Z}_{N_k}$ for $N_1, \ldots, N_k \in \mathbb{Z}^+$ (*i.e.*, positive integers), and $\mathbb{Z}_N$ is the additive group of integers modulo N.

The *Fourier transform* of f is the function $\widehat{f}\colon G \to \mathbb{C}$ defined for each $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ by

$$\widehat{f}(\alpha) \stackrel{def}{=} \frac{1}{|G|} \sum_{(x_1,\ldots,x_k)\in G} \left[f(x_1,\ldots,x_k) \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j} \right]$$

where $\omega_{N_j} = e^{i\frac{2\pi}{N_j}}$ is a primitive root of unity of order N_j. For any $\alpha \in G$, $val_\alpha \in \mathbb{C}$ and $\tau, \varepsilon \in [0,1]$, we say that α is a *τ-significant Fourier coefficient* iff $\left|\widehat{f}(\alpha)\right|^2 \geq \tau \|f\|_2^2$, and we say that val_α is an *ε-approximation for $\widehat{f}(\alpha)$* iff $\left|val_\alpha - \widehat{f}(\alpha)\right| < \varepsilon$.

[1] For readers more accustomed to vector notation, we remark that there is a simple correspondence between vector and functional notation. For example, a one dimensional signal $(v_1, \ldots, v_N) \in \mathbb{C}^N$ corresponds to the function $f\colon \mathbb{Z}_N \to \mathbb{C}$ defined by $f(i) = v_i$ for all $i = 1, \ldots, N$. Likewise, a two dimensional signal $M \in \mathbb{C}^{N_1 \times N_2}$ corresponds to the function $f\colon \mathbb{Z}_{N_1} \times \mathbb{Z}_{N_2} \to \mathbb{C}$ defined by $f(i,j) = M_{ij}$ for all $i = 1, \ldots, N_1$ and $j = 1, \ldots, N_2$.

Problem 1 (SFT)**.**
INPUT: *Integers* $N_1, \ldots, N_k \geq 2$ *specifying the group* $G = \mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$*, a threshold* $\tau \in (0,1)$*, an approximation parameter* $\varepsilon \in (0,1)$*, and oracle access*[2] *to* $f \colon G \to \mathbb{C}$*.*
OUTPUT: *A list of all* τ*-significant Fourier coefficients of* f *along with* ε*-approximations for them.*

2 KEY RESULTS

The key result of this entry is an algorithm solving the SFT problem which is much much faster than algorithms for computing the entire Fourier transform. For example, for f a Boolean function over $\mathbb{Z}_N$, the running time of this algorithm is $\log N \cdot poly\left(\log\log N, \frac{1}{\tau}, \frac{1}{\varepsilon}\right)$, in contrast to the $\Theta(N \log N)$ running time of the FFT algorithm. This algorithm is named the SFT algorithm.

Theorem 1 (SFT algorithm [2])**.** *There is an algorithm solving the SFT problem with running time* $\log|G| \cdot poly\left(\log\log|G|, \frac{\|f\|_\infty}{\|f\|_2}, \frac{1}{\tau}, \frac{1}{\varepsilon}\right)$ *for* $|G| = \prod_{j=1}^{k} N_j$ *the cardinality of* G*.*

Remarks.

1. The above result extends to functions f over any finite abelian group G, as long as the algorithm is given a description of G by its generators and their orders [2].
2. The SFT algorithm reads at most $\log|G| \cdot poly\left(\log\log|G|, \frac{\|f\|_\infty}{\|f\|_2}, \frac{1}{\tau}, \frac{1}{\varepsilon}\right)$ out of the $|G|$ values of the signal.
3. The SFT algorithm is non adaptive, that is, oracle queries to f are independent of the algorithm's progress.
4. The SFT algorithm is a probabilistic algorithm having a small error probability, where probability is taken over the internal coin tosses of the algorithm. The error probability can be made arbitrarily small by standard amplification techniques.

The SFT algorithm of [2] follows a line of works solving the SFT problem for restricted function classes. Goldreich and Levin [9] gave an algorithm for Boolean functions over the group $\mathbb{Z}_2^k = \{0,1\}^k$. The running time of their algorithm is polynomial in k, $\frac{1}{\tau}$ and $\frac{1}{\varepsilon}$. Mansour [11] gave an algorithm for complex functions over groups $G = \mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$ provided that $N_1, \ldots, N_k$ are powers of two. The running time of his algorithm is polynomial in $\log|G|, \log\left(\max_{\alpha \in G}\left|\widehat{f}(\alpha)\right|\right), \frac{1}{\tau}$ and $\frac{1}{\varepsilon}$. Gilbert *et.al.* [6] gave an algorithm for complex functions over the group $\mathbb{Z}_N$ for any positive integer N. The running time of their algorithm is polynomial in $\log N$, $\log\frac{\max_{x \in \mathbb{Z}_N} f(x)}{\min_{x \in \mathbb{Z}_N} f(x)}$, $\frac{1}{\tau}$ and $\frac{1}{\varepsilon}$. Akavia *et.al.* [2] gave an algorithm for complex functions over any finite abelian group. The latter [2] improves on [6] even when restricted to functions over $\mathbb{Z}_N$ in achieving $\log N \cdot poly(\log\log N)$ rather $poly(\log N)$ dependency on N. Subsequent works [7] improved the dependency of [6] on τ and ε.

3 APPLICATIONS

We survey applications of the SFT algorithm [2] in the areas of computational learning theory, coding theory, cryptography, and algorithms.

[2] We say that an algorithm is given *oracle access* to a function f over G, if it can request and receive the value $f(x)$ for any $x \in G$ in unit time.

3.1 Applications in Computational Learning Theory

A common task in computational learning is to find a hypothesis h approximating a function f, when given only samples of the function f. Samples may be given in a variety of forms, *e.g.*, via oracle access to f. We consider the following variant of this learning problem: f and h are complex functions over a finite abelian group $G = \mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$, the goal is to find h such that $\|f - h\|_2^2 \leq \gamma \|f\|_2^2$ for $\gamma > 0$ an approximation parameter, and samples of f are given via oracle access.

A straightforward application of the SFT algorithm gives an efficient solution to the above learning problem, provided that there is a small set $\Gamma \subseteq G$ s.t. $\sum_{\alpha \in \Gamma} \left|\widehat{f}(\alpha)\right|^2 > (1 - \frac{\gamma}{3})\|f\|_2^2$. The learning algorithm operates as follows. It first runs the SFT algorithm to find all $\alpha = (\alpha_1, \ldots, \alpha_k) \in G$ that are $\frac{\gamma}{|\Gamma|}$-significant Fourier coefficients of f along with their $\frac{\gamma}{|\Gamma| \|f\|_\infty}$-approximations val_α, and then returns the hypothesis

$$h(x_1, \ldots, x_k) \stackrel{def}{=} \sum_{\alpha \text{ is } \gamma/|\Gamma|\text{-significant}} val_\alpha \cdot \prod_{j=1}^{k} \omega_{N_j}^{\alpha_j x_j}$$

This hypothesis h satisfies that $\|f - h\|_2^2 \leq \gamma \|f\|_2^2$. The running time of this learning algorithm and the number of oracle queries it makes is polynomially bounded by $\log |G|$, $\|f\|_\infty / \|f\|_2$, $|\Gamma| \, \|f\|_\infty / \gamma$.

Theorem 2. *Let $f \colon G \to \mathbb{C}$ be a function over $G = \mathbb{Z}_{N_1} \times \ldots \times \mathbb{Z}_{N_k}$, and $\gamma > 0$ an approximation parameter. Denote $t = \min \left\{ |\Gamma| \mid \Gamma \subseteq G \text{ s.t. } \sum_{\alpha \in \Gamma} \left|\widehat{f}(\alpha)\right|^2 > (1 - \frac{\gamma}{3})\|f\|_2^2 \right\}$. There is an algorithm that given $N_1, \ldots, N_k$, γ, and oracle access to f, outputs a (short) description of $h \colon G \to \mathbb{C}$ s.t. $\|f - h\|_2^2 < \gamma \|f\|_2^2$. The running time of this algorithm is $\log |G| \cdot poly(\log \log |G|, \|f\|_\infty / \|f\|_2, t \|f\|_\infty / \gamma)$.*

More examples of function classes that can be efficiently learned using our SFT algorithm are given in [3].

3.2 Applications in Coding Theory

Error correcting codes encode messages in a way that allows *decoding*, that is, recovery of the original message, even in the presence of noise. When the noise is very high, unique decoding may be infeasible, nevertheless it may still be possible to *list decode*, that is, to find a short list of messages containing the original message. Codes equipped with an efficient list decoding algorithm have found many applications (see [10] for a survey).

Formally, a binary code is a subset $\mathcal{C} \subseteq \{0,1\}^*$ of *codewords* each encoding some message. We denote by $C_{N,x} \in \{0,1\}^N$ a codeword of length N encoding a message x. The *normalized Hamming distance* between a codeword $C_{N,x}$ and a received word $w \in \{0,1\}^N$ is $\Delta(C_{N,x}, w) \stackrel{def}{=} \frac{1}{N} |\{ i \in \mathbb{Z}_N \mid C_{N,x}(i) \neq w(i) \}|$ where $C_{N,x}(i)$ and $w(i)$ are the i-th bits of $C_{N,x}$ and w, respectively. Given $w \in \{0,1\}^N$ and a noise parameter $\eta > 0$, the *list decoding* task is to find a list of all messages x such that $\Delta(C_{N,x}, w) < \eta$. The received word w may be given explicitly or implicitly; we focus on the latter where oracle access to w is given.

Goldreich and Levin [9] gave a list decoding algorithm for Hadamard codes, using in a cruicial way their algorithm solving the SFT problem for functions over the Boolean cube.

The SFT algorithm of [2] for the case of functions over Z_N is a key component in a list decoding algorithm given by Akavia *et.al.* [2]. This list decoding algorithm is applicable to a large class of codes. For example, it is applicable to the code $\mathcal{C}^{msb} = \{C_{N,x} \colon \mathbb{Z}_N \to \{0,1\}\}_{x \in \mathbb{Z}_N^*, N \in \mathbb{Z}^+}$ whose codewords are $C_{N,x}(j) = msb_N(j \cdot x \mod N)$ for $msb_N(y) = 1$ iff $y \geq N/2$ and $msb_N(y) = 0$ otherwise. More generally, this list decoding algorithm is applicable to any *Multiplication code* $\mathcal{C}^{\mathcal{P}}$

for $\mathcal{P}$ a family of *balanced* and *well concentrated* functions, as defined below. The running time of this list decoding algorithm is polynomial in $\log N$ and $1/(1-2\eta)$, as long as $\eta < \frac{1}{2}$.

Abstractly, the list decoding algorithm of [2] is applicable to any code that is "balanced", "well concentrated" and "recoverable", as defined next. A code is *balanced* if $\Pr_{j\in\mathbb{Z}_N}[C_{N,x}(j)=0] = \Pr_{j\in\mathbb{Z}_N}[C_{N,x}(j)=1]$ for every codeword $C_{N,x}$. A code is *well concentrated* if its codewords can be approximated by a small number of significant coefficients in their Fourier representation, and those Fourier coefficients have small gcd with N. A code is *recoverable* if there is an efficient algorithm mapping each Fourier coefficient α to a short list of codewords for which α is a significant Fourier coefficient.

The key property of concentrated codes is that received words w share a significant Fourier coefficient with all close codewords $C_{N,x}$. The high level structure of the list decoding algorithm of [2] is therefore as follows. First it runs the SFT algorithm to find all significant Fourier coefficients α of the received word w. Second for each such α, it runs the recovery algorithm to find all codewords $C_{N,x}$ for which α is significant. Finally, it outputs all those codewords $C_{N,x}$.

Definition 1 (Multiplication Codes [2]). *Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of functions. We say that $\mathcal{C}^{\mathcal{P}} = \{C_{N,x}\colon \mathbb{Z}_N \to \{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ is a* multiplication code *for $\mathcal{P}$ if for every $N\in\mathbb{Z}^+$ and $x\in\mathbb{Z}_N^*$, the encoding $C_{N,x}\colon \mathbb{Z}_N\to\{0,1\}$ of x is defined by*

$$C_{N,x}(j) = P(j\cdot x \bmod N)$$

Definition 2 (Well concentrated [2]). *Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N\to\mathbb{C}\}_{N\in\mathbb{Z}^+}$ be a family of functions. We say that $\mathcal{P}$ is* well concentrated *if $\forall N\in\mathbb{Z}^+, \gamma>0$, $\exists \Gamma\subseteq\mathbb{Z}_N$ s.t. (i) $|\Gamma| \le poly(\log N/\gamma)$, (ii) $\sum_{\alpha\in\Gamma}\left|\widehat{P}_N(\alpha)\right|^2 \ge (1-\gamma)\|P_N\|_2^2$, and (iii) for all $\alpha\in\Gamma$, $gcd(\alpha,N)\le poly(\log N/\gamma)$ (where $gcd(\alpha,N)$ is the greatest common divisor of α and N).*

Theorem 3 (List decoding [2]). *Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N\to\{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of efficiently computable[3], well concentrated and balanced functions. Let $\mathcal{C}^{\mathcal{P}} = \{C_{N,x}\colon \mathbb{Z}_N\to\{0,1\}\}_{x\in\mathbb{Z}_N^*, N\in\mathbb{Z}^+}$ be the multiplication code for $\mathcal{P}$. Then there is an algorithm that, given $N\in\mathbb{Z}_N^+$, $\eta<\frac{1}{2}$ and oracle access to $w\colon\mathbb{Z}_N\to\{0,1\}$, outputs all $x\in\mathbb{Z}_N^*$ for which $\Delta(C_{N,x},w)<\eta$. The running time of this algorithm is polynomial in $\log N$ and $1/(1-2\eta)$.*

Remarks.

1. The requirement that $\mathcal{P}$ is a family of *efficiently computable* functions can be relaxed. It suffices to require that the list decoding algorithm receives or computes a set $\Gamma\subseteq\mathbb{Z}_N$ with properties as specified in Definition 2.

2. The requirement that $\mathcal{P}$ is a family of *balanced* functions can be relaxed. Denote $bias(\mathcal{P}) = \min_{b\in\{0,1\}}\inf_{N\in\mathbb{Z}^+}\Pr_{j\in\mathbb{Z}_N}[P_N(j)=b]$. Then the list decoding algorithm of [2] is applicable to $\mathcal{C}^{\mathcal{P}}$ even when $bias(\mathcal{P})\ne\frac{1}{2}$, as long as $\eta < bias(\mathcal{P})$.

3.3 Applications in Cryptography

Hard-core predicates for one-way functions are a fundamental cryptographic primitive, which is central for many cryptographic applications such as pseudo-random number generators, semantic secure encryption, and cryptographic protocols. Informally speaking, a Boolean predicate P is a *hard-core predicate* for a function f if $P(x)$ is easy to compute when given x, but hard to guess with a non-negligible advantage beyond 50% when given only $f(x)$. The notion of hardcore predicates

[3] $\mathcal{P} = \{P_N\colon\mathbb{Z}_N\to\{0,1\}\}_{N\in\mathbb{Z}^+}$ is a family of *efficiently computable* functions if there is algorithm that given any $N\in\mathbb{Z}^+$ and $x\in\mathbb{Z}_N$ outputs $P_N(x)$ in time $poly(\log N)$.

was introduced by Blum and Micali [4]. Goldreich and Levin [9] showed a randomized hardcore predicate for any one-way function, using in a crucial way their algorithm solving the SFT problem for functions over the Boolean cube.

Akavia *et.al.* [2] introduce a unifying framework for proving that a predicate P is hard-core for a one-way function f. Applying their framework they prove for a wide class of predicates – *segment predicates*– that they are hard-core predicates for various well known candidate one-way functions. Thus showing new hard-core predicates for well known one-way function candidates as well as reproving old results in an entirely different way.

Elaborating on the above, a *segment predicate* is any assignment of Boolean values to an arbitrary partition of $\mathbb{Z}_N$ into $poly(\log N)$ segments, or dilations of such an assignment. Akavia *et.al.* [2] prove that any segment predicate is hard-core for any one-way function f defined over $\mathbb{Z}_N$ for which, for a non-negligible fraction of the x's in $\mathbb{Z}_N$, given $f(x)$ and y, one can efficiently compute $f(xy)$ (where xy is multiplication in $\mathbb{Z}_N$). This includes the following functions: the *exponentiation* function $EXP_{p,g}\colon \mathbb{Z}_p \to \mathbb{Z}_p^*$ defined by $EXP_{p,g}(x) = g^x \bmod p$ for each prime p and a generator g of the group $\mathbb{Z}_p^*$; the *RSA* function $RSA\colon \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $RSA(x) = e^x \bmod N$ for each $N = pq$ a product of two primes p, q, and e co-prime to N; the *Rabin* function $Rabin\colon \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by $Rabin(x) = x^2 \bmod N$ for each $N = pq$ a product of two primes p, q; and the *elliptic curve log* function defined by $ECL_{a,b,p,Q} = xQ$ for each elliptic curve $E_{a,b,p}(Z_p)$ and Q a point of high order on the curve.

The SFT algorithm is a central tool in the framework of [2]: Akavia *et.al.* take a list decoding methodology, where computing a hard-core predicate corresponds to computing an entry in some error correcting code, predicting a predicate corresponds to access to an entry in a corrupted codeword, and the task of inverting a one-way function corresponds to the task of list decoding a corrupted codeword. The codes emerging in [2] are multiplication codes (see Definition 1 above), which are list decoded using the SFT algorithm.

Definition 3 (Segment predicates [2])**.** *Let $\mathcal{P} = \{P_N\colon \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ be a family of predicates that are non-negligibly far from constant*[4]*.*

- *We say that P_N is a* basic t-segment predicate *if $P_N(x+1) \neq P_N(x)$ for at most t x's in $\mathbb{Z}_N$.*
- *We say that P_N is a* t-segment predicate *if there exist a basic t-segment predicate P' and $a \in \mathbb{Z}_N$ which is co-prime to N s.t. $\forall x \in \mathbb{Z}_N, P_N(x) = P'(x/a)$.*
- *We say that $\mathcal{P}$ is a family of* segment predicates *if $\forall N \in \mathbb{Z}^+$, P_N is a $t(N)$-segment predicate for $t(N) \leq poly(\log N)$.*

Theorem 4 (Hardcore predicates [2])**.** *Let $\mathcal{P}$ be a family of segment predicates. Then, $\mathcal{P}$ is hard-core for $RSA, Rabin, EXP, ECL$, under the assumption that these are one-way functions.*

3.4 Application in Algorithms

Our modern times are characterized by information explosion incurring a need for faster and faster algorithms. Even algorithms classically regarded efficient —such as the FFT algorithm with its $\Theta(N \log N)$ complexity— are often too slow for data-intensive applications, and linear or even sub-linear algorithms are imperative. Despite the vast variety of fields and applications where algorithmic challenges arise, some basic algorithmic building blocks emerge in many of the existing algorithmic solutions. Accelerating such building blocks can therefore accelerate many existing algorithms. One of these recurring building blocks is the Fast Fourier Transform (FFT) algorithm. The SFT algorithm offers a great efficiency improvement over the FFT algorithm for applications

[4] A family of functions $\mathcal{P} = \{P_N\colon \mathbb{Z}_N \to \{0,1\}\}_{N\in\mathbb{Z}^+}$ is *non-negligibly far from constant* if $\forall N \in \mathbb{Z}^+$ and $b \in \{0,1\}$, $\Pr_{j\in\mathbb{Z}_N}[P_N(j) = b] \leq 1 - poly(1/\log N)$.

where it suffices to deal only with the significant Fourier coefficients. In such applications, replacing the FFT building block with the SFT algorithm accelerates the $\Theta(N \log N)$ complexity in each application of the FFT algorithm to $poly(\log N)$ complexity [1]. Lossy compression is an example of such an application [1, 5, 8]. Let us elaborate. A central component in several transform compression methods (*e.g.*, JPEG) is to first apply Fourier (or Cosine) transform to the signal, and then discard many of its coefficients. To accelerate such algorithms —instead of computing the entire Fourier (or Cosine) transform— the SFT algorithm can be used to directly approximate only the significant Fourier coefficients. Such an accelerated algorithm achieves compression guarantee as good as the original algorithm (and possibly better), but with running time improved to $poly(\log N)$ in place of the former $\Theta(N \log N)$.

4 CROSS REFERENCES

Abelian Hidden Subgroup Problem, Decoding Hadamard Codes and Learning Heavy Fourier Coefficients of Boolean Functions, Learning Constant-depth Circuits, Learning DNF Formulas, Learning with Malicious Noise, List Decoding near Capacity: Folded RS Codes, PAC Learning, Statistical Query Learning

5 RECOMMENDED READING

[1] A. Akavia and S. Goldwasser, *Manuscript submitted as an NSF grant, awarded 2005 CCF-0514167.*

[2] A. Akavia, S. Goldwasser, and S. Safra, *Proving hard-core predicates using list decoding*, in Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS'03), IEEE Computer Society, 2003, pp. 146–157.

[3] A. Atici and R. A. Servedio, *Learning unions of $\omega(1)$-dimensional rectangles.*, in ALT, 2006, pp. 32–47.

[4] M. Blum and S. Micali, *How to generate cryptographically strong sequences of pseudo-random bits*, SIAM Journal on Computing, 13 (1984), pp. 850–864.

[5] G. Cormode and S. Muthukrishnan, *Combinatorial algorithms for compressed sensing*, in Structural Information and Communication Complexity, 13th International Colloquium, SIROCCO 2006, Chester, UK, July 2-5, 2006, Proceedings, 2006, pp. 280–294.

[6] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss, *Near-optimal sparse fourier representations via sampling*, in Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, ACM Press, 2002, pp. 152–161.

[7] A. C. Gilbert, S. Muthukrishnan, and M. J. Strauss, *Improved time bounds for near-optimal sparse fourier representation via sampling*, in In Proceedings of SPIE Wavelets XI, San Diego, CA, 2005, 2005.

[8] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, *One sketch for all: Fast algorithms for compressed sensing*, in 39th ACM Symposium on Theory of Computing (STOC'07). To appear.

[9] O. Goldreich and L. Levin, *A hard-core predicate for all one-way functions*, in 27th ACM Symposium on Theory of Computing (STOC'89), 1989.

[10] Madhu Sudan, *List decoding: algorithms and applications*, SIGACT News, 31 (2000), pp. 16–27.

[11] Y. Mansour, *Randomized interpolation and approximation of sparse polynomials*, SIAM Journal on Computing, 24 (1995), pp. 357–368.